Template based code generation of Modelica building energy simulation models

Christoph Nytsch-Geusen[1] Alexander Inderfurth[1] Werner Kaul[1]
Katharina Mucha[1] Jörg Rädler[1] Matthis Thorade[1] Carles Ribas Tugores[1]

[1]Institut für Architektur und Städtebau, Berlin University of the Arts, Germany, nytsch@udk-berlin.de

Abstract

This contribution describes an approach for a template based code generation for different detailed Modelica models for building energy simulation (BES).

The information from several data sources, which describe the building geometry, the building construction, the building location and the building itself, is used to fill a building data model. This intermediate data structure is still independent of a certain building simulation tool.

A new developed tool for template based code generation (CoTeTo) uses the building data model and combines it with a set of different code generators, which are able to generate Modelica building models with a different level of detail: Strong simplified low-order building models for district energy simulation with a large population of buildings, more advanced multi-zone building models for building energy simulation and 3D space resolved room models for a detailed indoor climate analysis.

Three case studies for the mentioned building model types demonstrate the code generation approach.

Keywords: building energy simulation, adapted model level of detail, Modelica code generation

1 Introduction

The generation of machine-readable code usually combines static and dynamic data sources. The static part describes the keywords and syntactical requirements of a computer language and builds a static framework while the dynamic part injects real values and structures from the runtime environment of the code-generating application or from an external data source. In the simplest case the application uses some (potentially nested) print()-like statements. This approach has some limitations because even the smallest change in the output format requires access to the source code of the application, programming skills and potentially large compile cycles.

With the rise of dynamic web-sites a more flexible technology was widely used and much improved: the template engines. Such an engine is a program library linked into an application, but the process of the code generation is controlled by external text files. These template files embed simple control structures and placeholders in normal text and can be easily edited. The concept is similar to the serial letter function in word processing applications.

The idea of code generation for Modelica BES libraries was first applied within the EnEff BIM project. In this project the structured data of an IFC files were used for the automatic generation of Modelica system models, consisting of a HVAC sub-model and a strong simplified building model (for more details see Thorade et al., 2015).

This contribution is focused on code generation for Modelica building energy models with different levels of detail. Important information for the code generation are the building geometries, the building topologies, the building constructions, the building locations and the behavior of the building occupants.

2 Template based code generation

A general approach for code generation of BES models has to consider the heterogeneous data formats (data sources) in the building industry sector and should be able to generate models with a different level of detail, which fits to the question of the simulation analysis.

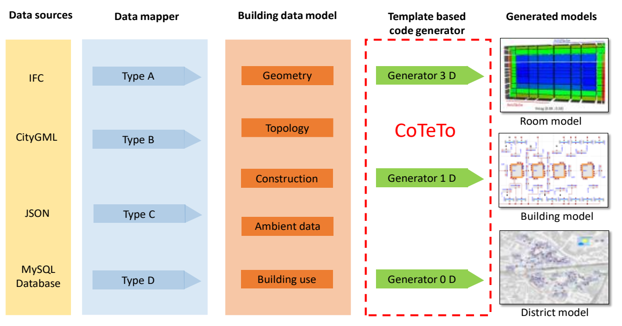


Figure 1. Template based code generation of BES models with a different level of detail

A set of data mappers transform the input data into a common building data model. Dependent on the present information within this data model one or more template based code generators can produce Modelica BES models for room simulation, building simulation or district simulation (compare with Figure 1).

2.1 Data sources

In the building industry sector, there are different data sources and data formats available, which can satisfy the needs of the building energy simulation domain. On the scale of single buildings, the IFC-Format in the version IFC2x3 (IFC2x3, 2017) can be used and in near future also the version IFC4 (IFC4, 2017). This format represents the digital building model in a well-structured form (the entire building, several spaces, walls, windows etc.) in combination with a precise description of the building geometry. Most of the architecture CAD programs can export the IFC data format. It fits perfect to the structure of a multi-zone-building model (building model, thermal zones, building components) and the precise geometrical data also allows the parameterization of spatial resolved room models.

On the scale of city districts the CityGML format (CityGML, 2017) and the GeoJSON format (GeoJSON, 2017) can deliver the necessary building parameter for district energy models. Normally, GIS programs are able to export one or both of these data formats with simplified building geometries, which fits to the reduced parameter sets of the low-order building models on the district model scale. In this case, the challenge consists in the data acquisition of huge populations of buildings and not for a single building (Kaul et al., 2014).

In special cases, building parameter sets are also available in data base formats, e.g. MySQL (Inderfurth et al., 2017).

2.2 Data mapper

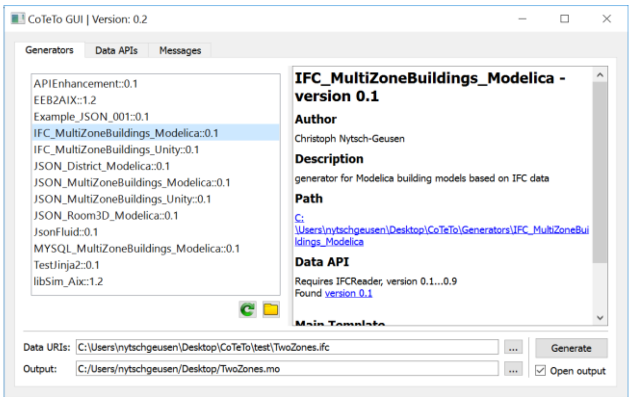
A data mapper is a specialized software module, which is able to map a certain data source file format to the format independent building data model (see paragraph 2.3). Two different data mappers were realized based on Python up to now: the first data mapper can be used for 1-dim. multi-zone-building simulation and 3-dim. room simulation and uses the IFC format as the data input. The Python bindings of the IfcOpenShell-library (IfcOpenShell, 2017) are used to read the IFC-files and Python bindings of the OpenCascade-library (pythonOCC, 2017) are used to transform in a second step the geometrical and the topology data in a manner, that they can be stored in the building data model. The second data mapper was implemented for district energy simulation and can read the GeoJSON-format. A third data mapper for information input from SQL data bases is under development.

2.3 Building data model

The building data model holds all the information, which is necessary for the Modelica code generation. This includes the data for the building geometry (full geometrical description or simplified geometry), the building topology (substructure of a building in thermal zones), the used construction types (multi-layer definitions), the definition of the building ambient data (location, weather data) and the type of building use (e.g. air change rates, set temperatures for heating and cooling etc.). The building data model itself is independent of the type of the data sources (but it has functions for setting building parameters from data sources) and also on the type of the code generator (different code generators use the same function to get building parameters from the building data model).

2.4 CoTeTo

To automate some of the required steps for the generation and parametrization of Modelica code a software tool (**Co**de **Te**mplating **To**ol) was developed in the context of the EnEff-BIM project (Thorade et al, 2015). CoTeTo (CoTeTo, 2017) comes with an open source license and can be download from GitHub (https://github.com/UdK-VPT/CoTeTo). It includes pluggable input, filter and output components that cover the process of data acquisition, preprocessing and output using a template system. CoTeTo is implemented in Python and can be used standalone or as a library imported in Python applications. A command line interface is provided for interactive usage or inclusion in shell scripts. A GUI based on PyQt4 (PyQt4, 2017) can be started as an application (see Figure 2Figure 1) or included in PyQ4-based applications as a widget.



APIEnhancement::0.1
EEB2AIX::1.2
Example_JSON_001::0.1
IFC_MultiZoneBuildings_Modelica::0.1
IFC_MultiZoneBuildings_Unity::0.1
JSON_District_Modelica::0.1
JSON_MultiZoneBuildings_Modelica::0.1
JSON_MultiZoneBuildings_Unity::0.1
JSON_Room3D_Modelica::0.1
JsonFluid::0.1
MYSQL_MultiZoneBuildings_Modelica::0.1
TestJinja2::0.1
libSim_Aix::1.2

IFC_MultiZoneBuildings_Modelica - version 0.1

Author

Christoph Nytsch-Geusen

Description

generator for Modelica building models based on IFC data

Path

C:\Users\nytschgeusen\Desktop\CoTeTo\Generators\IFC_MultiZoneBuildings_Modelica

Data API

Requires IFCReader, version 0.1...0.9
Found version 0.1

Data URIs: C:\Users\nytschgeusen\Desktop\CoTeTo\test\TwoZones.ifc
Output: C:/Users/nytschgeusen/Desktop/TwoZones.mo
[✓] Open output

Figure 2. CoTeTo GUI for template based code generation

CoTeTo uses the Mako template engine (Mako, 2017) for the code generation step, but an experimental interface to the Jinja2 engine (Jinja2, 2017) is implemented as well.

2.5 Generators

CoTeTo documents (called generators) can be easily edited and shared without deep programming knowledge. A generator is stored in a folder structure or a zip file containing plain text files. The idea of a generator is to include all parts necessary to generate the code for a defined target (like a certain Modelica buildings library) form a defined source (like a special file format or database structure).

A generator depends on a so-called input API, which is defined in a Python module. Some standard input APIs are included in CoTeTo (CSV, JSON, XML, …), but generators can define own input modules. Between the data input and the output templates filter functions can be called to preprocess the data structure. These Python functions are defined in the generator.

The CoTeTo framework handles this conversion process completely data-agnostic, the structure and format of the data objects is defined by the input APIs, generators and filter functions only.

2.6 Adaption to the BuildingSystems library

Based on the CoTeTo framework three code generators for the Modelica BuildingSystems library (http://www.modelica-buildingsystems.de) were implemented. This library is being developed for the dynamic simulation of the energetic behavior of single rooms, multi-zone buildings or entire city districts (Nytsch-Geusen et al., 2016). The simulation models of the library describe the dynamic energy balance of the building envelope under consideration of the building geometry, the thermal properties of the building construction, the ambient climate and the user behavior. As the Modelica library IDEAS, AIX Lib and Buildings, the BuildingSystems library uses as a core the same Annex 60 Library, which was developed as a common project from the authors of the four mentioned libraries in the Annex 60 project (Wetter et al., 2015).

The predefined components of the BuildingSystems library such as air volumes models, building construction models, wall and window models, zone models, low-order building models or ambient models (compare Figure 3, Figure 6 and Figure 10) are the base for the generated Modelica code. These model classes include the physical description (energy and mass balances, empirical equations etc.) and are instantiated and parameterized by the code generator using the information, which is stored in the building data model.

The following code shows as an example the Mako code, which generates the Modelica records for the definition of all multi-layered opaque constructions of a building model:

```
% for con in constructions:
record ${con.name}
 extends OpaqueThermalConstruction(
  nLayers=${con.nLayers},
  thickness={
% for value in con.thickness:
  ${value}${',' if not loop.last else ''}
% endfor
  },
  material={
  % for value in con.material:
  ${value}()${',' if not loop.last else ''}
  % endfor
 });
end ${con.name};
% endfor
```

Based on the stored information in the building data model the code generator generates for example the code for three different building constructions:

```
record ConstructionFacade
 extends OpaqueThermalConstruction(
 nLayers=4,
 thickness={0.015,0.2,0.15,0.02},
 material={
  HighGradePlaster(),
  Concrete(),
  ExpandedPolystyrene(),
  HighGradePlaster()});
end ConstructionFacade;

record ConstructionInnerWall
 extends OpaqueThermalConstruction(
 nLayers=3,
 thickness={0.015,0.12,0.015},
 material={
  HighGradePlaster(),
  Kalksandstein1800(),
  HighGradePlaster()});
end ConstructionInnerWall;

record ConstructionBottom
 extends OpaqueThermalConstruction(
 nLayers=3,
 thickness={0.02,0.06,0.2},
 material={
 Wood(),
 WoodFibreInsulation(),
 Concrete()});
end ConstructionBottom;
```

3 Case studies

The case studies shall demonstrate the general approach for template based Modelica code generation for building energy simulation. The examples address three different scales of building simulation: District modelling, multi-zone modelling and single room modelling.

3.1 City district

The first case study considers a city district in Berlin-Kreuzberg, which was designated by the Berlin city government as a redevelopment area (SenStadtWohn, 2016). In this context an analysis about the present energy efficiency of the building stock within this areal will be of interest. Because the whole district includes 144 buildings, the challenge for a district energy model, which could describe the present energy demand, consists in the data gathering of a huge parameter set (geometries, U-values etc.) for all buildings.

Data source: In the former research project Open eQuarter, a new layer-oriented geographic information system (GIS) based method was developed to obtain building sharp parameter data sets (Kaul et al., 2014). For this purpose, different city maps with information such as the building outlines, the number of stories, the building age in combination with a data base with U-values of the building elements were used, dependent on the building age (Loga et al., 2015). The open source GIS tool QGIS (QGOS, 2017) in combination with the

Open eQuarter plugin is able to export a GeoJSON file, which includes all the necessary building parameters (location, simplified building geometries, U-values) gained and calculated by the mentioned data sources. This GeoJSON file serves as the data input for the building data model in Figure 1.

Data mapper: In a first step the building data model takes the information through a data mapper from a GeoJSON file, which contains beside the mentioned building parameters also the building outlines for each building as polygon points. A python filter function calculates the centroids of these polygons to obtain the local placements of the building models within the district model. After this intermediate step all needed building data are stored in the building data model and can be used afterwards by the Modelica code generator.

Components: Two components of the BuidingSystems library are used for the code generation (see Figure 3). First, an ambient model, which describes the climate boundary condition of the city district, in particular the outside air temperature and the solar radiation on the building surfaces. Second, a low order building model (described in Nytsch-Geusen and Kaul, 2015), which is able to calculate the dynamic heating and cooling demand for an individual building with a small set of input parameters.

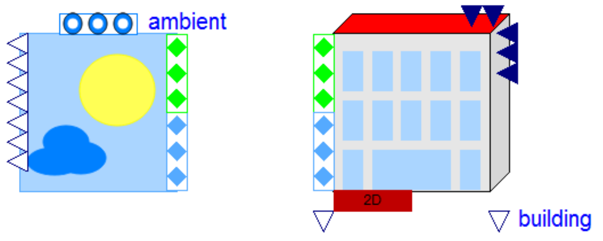


Figure 3. Components for district modelling.

Code generator: During the code generation the building centroids are used for component related annotations, which defines the graphical appearance of the individual building models on a realistic position. This is possible, because the positions of each individual building were gained from geo-referenced maps (compare with Figure 4). The excerpt of the generated code shows the instantiation and parameterization of the first two building models of the district, the ambient models and the connections between the ambient model and the two building models:

```
model DistrictModel
 extends Modelica.Icons.Example;
 Building1Zone0DDistrict building1(
  UValFac = 0.371,
  UValRoo = 0.269,
  UValGro = 0.4,
  UValWin = fill(1.575,4),
  fWin = 0.21,
  length = 8.127566,
  width = 5.318865,
  heightSto = 3.0,
  nSto = 4)
  annotation(Placement(transformation(
   extent={{0.0,0.0},{10.0,10.0}})));
 Building1Zone0DDistrict building2(
  UValFac = 1.83,
  UValRoo = 1.23,
  UValGro = 1.2,
  UValWin = fill(3.1,4),
  fWin = 0.294,
  length = 48.020794,
  width = 7.903955,
  heightSto = 3.0,
  nSto = 4)
  annotation(Placement(transformation(
  extent={{29.574,1.040},{19.574,11.040}})));
 ...
 Ambient ambient(
  nSurfaces = 720,
  weatherDataFile = WeatherDataFile_Berlin());
equation
 connect(ambient.toSurfacePorts[1:5],
  building1.toAmbientSurfacesPorts[1:5]);
 connect(ambient.toAirPorts[1:5],
  building1.toAmbientAirPorts[1:5]);
 connect(ambient.TAirRef, building1.TAirAmb);
 connect(ambient.xAir, building1.xAirAmb);
 connect(building1.airchange[1],airchange.y);
 connect(building1.T_setHeating[1],TSetHeating.y
 );
 connect(building1.T_setCooling[1],TSetCooling.y
 );
 ...
 connect(ambient.toSurfacePorts[6:10],
  building2.toAmbientSurfacesPorts[6:10]);
 connect(ambient.toAirPorts[6:10],
  building2.toAmbientAirPorts[6:10]);
 ...
end DistrictModel;
```

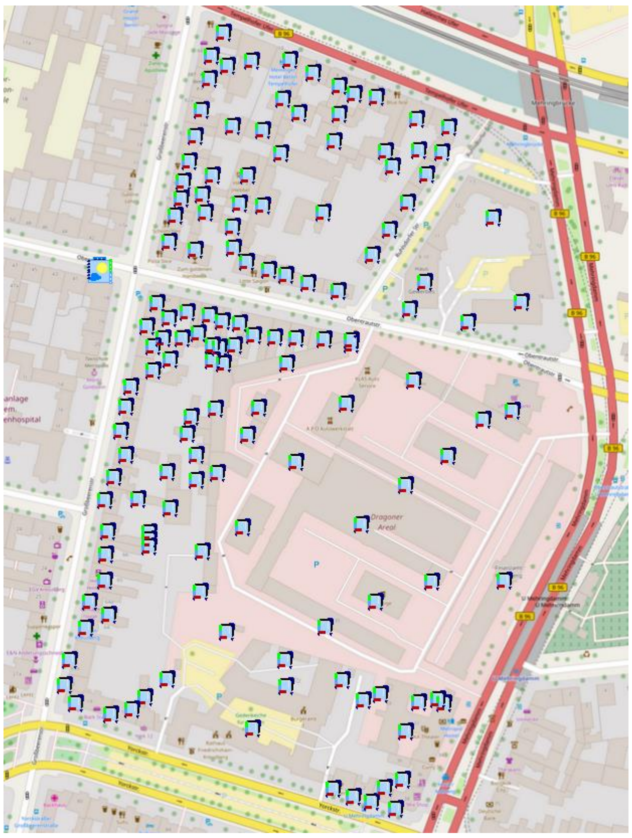


Figure 4. Generated Modelica district model with 144 low-order building models (the City map is taken from OpenStreetMap, https://www.openstreetmap.org)

3.2 Multi-zone building

The second case study demonstrates the code generation of a multi-zone building model (a storey of an office building) with thirteen thermals zones. It includes eight single office rooms, each of them with the same floor space, oriented to the North. Second, it has a bullpen with a large south oriented window and a smaller west oriented window. Beside the bullpen a conference room is attached, which has also a south oriented window. Further the story includes two sanitary rooms without windows and a corridor, which divides the north oriented by the south oriented rooms as a thermal buffer zone.

Data source: The building was constructed in Archicad 19 (see Figure 5) and afterwards exported as an IFC2x3 file. This model includes a precise description of the building geometry, topology and also the information about the layered construction of the building (used materials and the thicknesses of each layer).

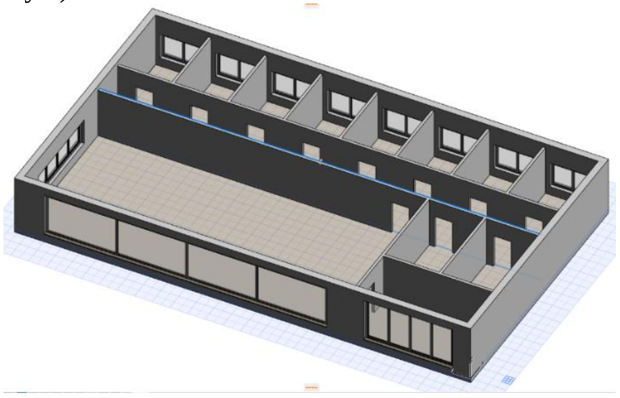


Figure 5. Building model, constructed in Archicad 19.

Data mapper: The data mapper reads the IFC file and analyses the building geometry and modifies if necessary the topology. For example, the south façade of the building is constructed in the CAD tool as one continuous element, but it has to be divided into two individual thermal wall models, because these models will have different thermal boundary conditions in a multi-zone building model. After this analysis the building data is stored in the building data model.

Components: Different models of the BuildingSystems library (opaque and transparent building element models, zone models, building template models etc. and again an ambient model) are used as the base for the code generation (see Figure 6).

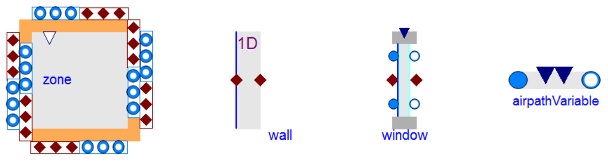


Figure 6. Components for multi-zone modelling.

Code generator: In this case study the stored information in the building data model is used twice: First for the generation of the Modelica code of the thermal multi-zone building model (see Figure 7) and second for a corresponding C# script, which is able to visualize the simulation results within a 3-dimensional building model (see Figure 8), based on Unity 5 (Nytsch-Geusen et al., 2017).

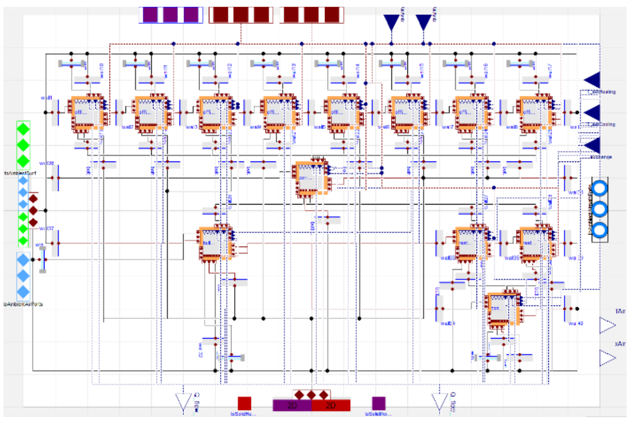


Figure 7. Generated Modelica multi-zone building model with 13 thermal zones.

The excerpt of the generated code shows the instantiation of the individual opaque and transparent building elements, thermal zones and their connections to a multi-zone building model (`model Building`). In the next step this "container class" is instantiated and connected on a higher level together with the ambient model to the Modelica system model (`model MultiZoneBuilding`):

```
model MultiZoneBuilding
 extends Modelica.Icons.Example;

 record ConstructionFacade
  extends OpaqueThermalConstruction(
   nLayers=4,
   thickness={0.015,0.2,0.15,0.02},
  ...

 model Building
  extends BuildingTemplate(
   nZones = 13,
   surfacesToAmbient(nSurfaces = 43),
   nSurfacesSolid = 13, ...);

  // building zones
  ZoneTemplateAirvolumeMixed office1(
   V=36.0,height=3.0,
   nConstructions1=8,...);
  ...
  ZoneTemplateAirvolumeMixed bullpen(
   V=450.0,height=3.0,
   nConstructions1=11,...);

  // constructions elements
  WallThermal1DNodes wall11(
   redeclare ConstructionFacade
   constructionData,
   angleDegAzi = 180.0,angleDegTil = 90.0,
   nInnSur = 1, AInnSur = {window2.A},
   height = 3.0,width = 3.0);
  ...
  Window window2(
   angleDegAzi = 180.0,angleDegTil = 90.0,
   height = 1.5,width = 2.5, UVal = ...);
  equation
   // construction elements <-> zones
   connect(wall11.toSurfacePort_1,
    office2.toConstructionPorts1[1]);
   connect(window2.toSurfacePort_1,
    office2.toConstructionPorts1[5]);
   ...
   // construction elements <-> ambient
   connect(window2.toSurfacePort_2,
    surfacesToAmbient.toConstructionPorts[5]);
   connect(wall11.toSurfacePort_2,
    surfacesToAmbient.toConstructionPorts[6]);
   ...
   // construction elements <-> ground
   connect(bottom1.toSurfacePort_2,
    surfacesToSolids.toConstructionPorts[1]);
   ...
  end Building;

  Building building(
   show_TSur = true,nSurfaces = 182,nZones = 13);
  Ambient ambient(
   nSurfaces = building.nSurfacesAmbient,
   weatherDataFile = WeatherDataFile_Berlin());
equation
  connect(ambient.toSurfacePorts,
   building.toAmbientSurfacesPorts);
  connect(ambient.toAirPorts,
   building.toAmbientAirPorts);
  connect(ambient.TAirRef, building.TAirAmb);
  connect(ambient.xAir, building.xAirAmb);
...
end MultiZoneBuilding;
```

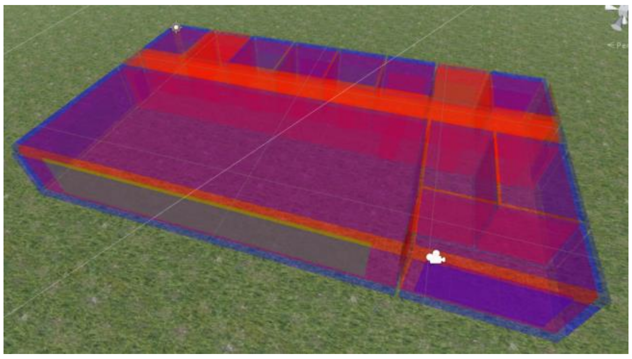


Figure 8. Generated multi-zone Unity building model for visualization of simulation results.

Figure 8 shows the visualization of the simulated surface temperatures of the multi-zone building model. The following code is an excerpt of the automatically generated C# script, which instantiates in Unity 5 this 3D visualization model:

```
using UnityEngine;
using System.Collections;

public class Surfaces : MonoBehaviour{
 public GameObject[] surfaces;
 private int nSur = 182;
 private Vector3 dirY = new Vector3(0,1,0);
 private Vector3 dy = new Vector3(0,0,0);
 private float[] rgb = new float[3];
 private float time = 0.0F;

 void Start(){
  sur = new GameObject[nSur];
  sur[0] = GameObject.CreatePrimitive(
   PrimitiveType.Cube);
  sur[0].name = "wall1_sur1";
  sur[0].transform.localScale =
   new Vector3(4.0F,0.01F,3.0F);
  sur[0].GetComponent<Renderer>().material=
   new Material(Shader.Find("Transparent/Diffuse"));
  sur[0].GetComponent<Renderer>().material.
   color = new Color(1, 0, 0, 0.3F);
  sur[0].transform.rotation =
   Quaternion.Euler(90.0F,90.0F,0.0F);
  sur[0].transform.position =
   new Vector3(0.0F,1.5F,-2.0F);
  sur[0].GetComponent<Collider>().enabled = false;
  dy = sur[0].transform.TransformDirection(dirY);
  sur[1] = GameObject.CreatePrimitive(
   PrimitiveType.Cube);
  sur[1].name = "wall1_sur2";
  ...
 }
 void Update(){
  time += 0.01F;
  float[] T_Surface = new float[]{
  // C# code for reading the simulation results
  // from the Modelica simulation
  ...
  }
  for (int i = 0; i < nSurfaces; i++){
   rgb = RGBMapper (T_Surface[i],10.0F,30.0F);
   sur[i].GetComponent<Renderer>().
    material.color=
   new Color(rgb[0],rgb[1],rgb[2],0.3F);}
  }
 }
```

3.3 Single room

The third case study for template based code generation was taken from the DFG Forschergruppe 1736 UCaHS (UCaHS, 2017). Within this project, the indoor climate of a patient room in a Berlin hospital (see Figure 9) was analyzed in detailed regarding the heat stress risk during hot summer weeks.

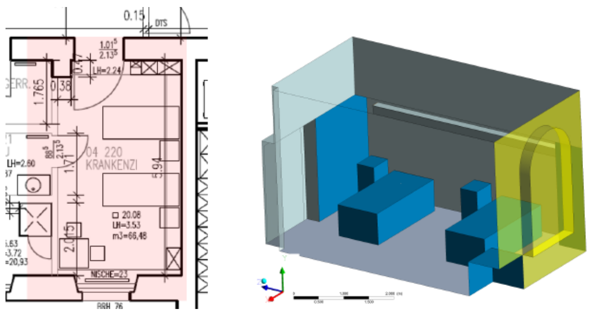


Figure 9. Floorplan and 3D model of the patient room.

For this purpose, a discretized room model in Modelica, a so called “zonal model”, which is based on a finite-volume-method and a simplified implementation of the Navier-Stokes equations was developed by Mucha (2017). A typical configuration of this room model includes between 300 to 500 air volume models, which are interconnected to each other by coupling models, which consider the friction between the air layers and the momentum transport. Caused by the high number of air volume elements and their necessary interconnections a manually failure free configuration of a room model, especially for non-box-shaped rooms would be nearly impossible.

Data source: At the moment the geometrical description of the 3-dim. room geometry inclusive its space discretization and also the physical parameter of the building construction are stored in a structuresd JSON file.

DataMapper: The data mapper reads the building parameter from the JSON file and stores it in the building data model.

Components: Figure 10 shows the components of the BuildungSystems library, which are used for the configuration of the space-discretized room model: an air volume model (energy and mass balance), a flow element model (friction calculation within the air), a heat conduction model (heat conduction within the air) and an interface model for the boundary condition of the room model (surface, wall and window models).

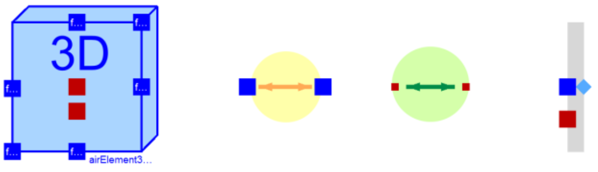


Figure 10. Components for room modelling.

Code generator: The code generator takes the information from the building data model and generates the Modelica code for the space discretized room model. This case study clearly demonstrates the advantage of the template based code generation approach. More than 500 air volume models have to be connected in three room coordinates with flow element models. In addition, different special cases have to be considered during the code generation process, for example the presence of furniture or the changing boundary condition models at the borders of the air space (e.g. a connection of a border air volume model with an adjacent wall or opening model).

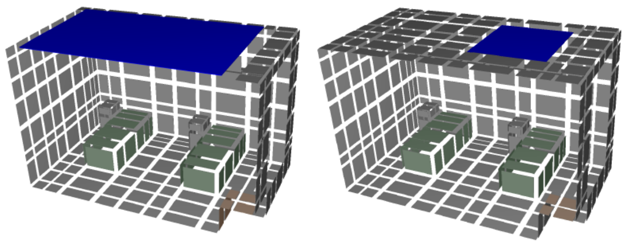


Figure 11. Generated discretized Modelica room model with 532 air volume models.

Figure 11 shows a variation of a generated room model of the patient room: one with a large cooling ceiling and one with a small cooling ceiling, which covers only the area of one of the patient beds. The correspondent adaptions in the building data model, before the code generation is repeated for the varied model are relative simple in comparison to manually changes in the generated code of the originally model.
The excerpt of the generated code exemplary shows the instantiation of two of the air volume elements, the flow and the heat conduction elements and the interconnections of the components to the 3-dimensional air flow model:

```
model Room
 ...
 FlowConnectionY floConY5710;

 ZoneHeatConductionY heaConY5710;

 AirElementThermal airEle6710(
  posX= vecX[10], posY= vecY[6], posZ= vecZ[7],
  T_start = T_inside,
  scalF = {scalX[10],scalY[6],scalZ[7]},
  enabled = false, BCwall_west = false,
  BCwall_east = true, BCwall_floor = false,
  BCwall_roof = false, BCwall_south = false,
  BCwall_north = true);
 ...
 FlowConnectionY floConY6710;

 ZoneHeatConductionY heaConY6710;

 AirElementThermal airEle7710(
  posX= vecX[10], posY= vecY[7], posZ= vecZ[7],
  T_start = T_inside,
  scalF = {scalX[10],scalY[7],scalZ[7]},
  enabled = false,BCwall_west = false,
  BCwall_east = true, BCwall_floor = false,
  BCwall_roof = true, BCwall_south = false,
  BCwall_north = true);
  ...
equation
  ...
  connect(floConX679.Port2, airEle6710.PortX1);
  connect(airEle679.PortHeatIntern,
   heaConX679.Port1);
  connect(heaConX679.Port2,
   airEle6710.PortHeatIntern);
  connect(airEle679.PortY2, heaConY679.Port1);
  connect(floConY679.Port2, airEle779.PortY1);
  connect(airEle679.PortHeatIntern,
   heaConY679.Port1);
  connect(heaConY679.Port2,
   airEle779.PortHeatIntern);
  connect(airEle779.PortX2, airEle779.Port1);
  ...
end Room;
```

3.4 Analysis and discussion

The three case studies are compared to each other with the help of benchmark values, e.g. the line of codes, the number of components or the number of connections within the generated system model (see Table 1).

Table 1. Comparison of the three case studies.

	District	**Building**	**Room**
Lines of code	2,904	1,544	15,985
Number of components	150	173	3713
Number of connections	1,008	544	10,982
Number of equations	435,765	40,434	132,712
Continuous time states	1,872	305	2,481
Time-varying variables	34,285	3,139	30,194

It can be stated, that building energy simulation analysis in Modelica usually leads to large system models. System models with 3,000 up to 16,000 lines of Modelica code cannot be manually configured failure free. The number of the components reaches from 150 to 3,713 and the number of connections from 544 to 10,982.
The generated models of Table 1 can be compiled and simulated without any problems by Dymola 2017 FD01. A test width a generated district model with more than 500 buildings illustrated the present limitations of the Modelica simulation tools: Dymola 2017 FD01 was not able to compile this large model, neither with a 64 bit compiler.
In the case of the district model, information from the GIS system can be used to generate a Modelica model which is able to display the real location of the individual buildings in the city map.
In the case of the multi-zone building model the input data can be used to generate consistent program code for two different purposes (Modelica and Unity code).
In the case of the room model, the code generator enables configuration of 3 dimensional models, which cannot be really modeled within a 2-dimensional graphical editor of a Modelica simulation tool.

4 Summary and Outlook

The described new approach for a template based code generation for Modelica building models was successfully applied to three different case studies on different room scales: district simulation, multi-zone building simulation and room simulation. A building data model, which stores the information in a structured and compact manner in combination with a template based code generator (CoTeTo), can avoid failures of manually written large Modelica system models.

In the next development step, the described Modelica code generators will be extended for special modelling cases. For this purposes, Mako code for conditional code generation will be introduced, which allows variations of generated components and connections within the Modelica system model.

The import of complex building or district data based on IFC or CityGML can be potentially incomplete or error prone. For this purpose, a graphical viewer incl. a consistency check shall be developed in future to obtain a more reliable base for the following code generation process.

Modelica simulator developers should improve their tools regarding the compiler technologies and also their numerical efficiency and flexibility. Especially large city district models, which can be easily generated from the GIS data with the described method, can address a lot of computer memory and potentially need a huge amount of numerical resources. In this context, the application of parallel computing technologies could improve the situation.

Acknowledgements

The research described in this paper is conducted within research project “EnEff BIM: Planung, Auslegung und Betriebsoptimierung von energieeffizienten Neu- und Bestandsbauten durch Modellierung und Simulation auf Basis von Bauwerkinformationsmodellen” funded by the Federal Ministry for Economic Affairs and Energy in Germany (reference: 03ET1177D).

References

CityGML. Exchange and storage of virtual 3D city models - http://www.citygml.org (last access on 2017 Jan 20).

CoTeTo - Code Templating Tool - https://github.com/UdK-VPT/CoTeTo (last access on 2017 Jan 20).

Alexander Inderfurth, Arda Karasu, Christoph Nytsch-Geusen, Claus Steffan. Architectural-Geometrical Simplification for Multi-Zone Building Models for Urban Refurbishment Projects. Accepted for Building Simulation 2017, 15th International Conference of IBPSA. San Francisco, August 2017.

GeoJSON. A format for encoding a variety of geographic data structures - http://geojson.org (last access on 2017 Jan 20).

PyQt4. Python bindings for the Qt application framework - https://riverbankcomputing.com/software/pyqt (last access on 2017 Jan 20)

Werner Kaul, Christoph Nytsch-Geusen, Phillip Wehage, and Michael Färber. Teilautomatisierte Akquise energetischer Gebäudedaten für die Quartiersanalayse und - simulation durch den Einsatz von Geo-Informations-Systemen (GIS). BAUSIM 2014 IBPSA Germany. Conference Proceedings. Aachen, September 2014.

Tobias Loga, Britta Stein, Nikolaus Diefenbach, and Rolf Born. Deutsche Wohngebäudetypologie. Beispielhafte Maßnahmen zur Verbesserung der Energieeffizienz von typischen Wohngebäuden, Institut Wohnen und Umwelt, Darmstadt / Germany, ISBN: 978-3-941140-47-9, 2015.

IFC2x3. IFC2x Edition 3 specification - http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html (last access on 2017 Jan 20)

IFC4. IFC4 specification - http://www.buildingsmart-tech.org/ifc/IFC4/final/html/ (last access on 2017 Jan 20)

IfcOpenShell. The open source IFC toolkit and geometry engine - http://ifcopenshell.org/python.html (last access on 2017 Jan 20)

Mako. Mako templates for python - http://www.makotemplates.org (last access on 2017 Jan 20)

Jinja2. Jinja2 (the python template engine) - http://jinja.pocoo.org (last access on 2017 Jan 20)

Christoph Nytsch-Geusen, and Werner Kaul. Generation of dynamic energetic district models from statistical relationships. 14th IBPSA Building Simulation Conference, Hyderabad, Conference Proceedings, December 2015.

Christoph Nytsch-Geusen, Christoph Banhardt, Alexander Inderfurth., Katharina Mucha, Jens Möckel, Jörg Rädler, Matthis Thorade, and Carles R. Tugores. BuildingSystems – Eine modular hierarchische Modell-Bibliothek zur energetischen Gebäude- und Anlagensimulation. BAUSIM

2016 IBPSA Germany, Conference Proceedings. Dresden, September 2016.

Christoph Nytsch-Geusen, Thaeba Ayubi, Jens Möckel, Jörg Rädler, Matthis Thorade. BuildingSystems_VR – A new approach for immersive and interactive building energy simulation. Accepted for Building Simulation 2017, 15th International Conference of IBPSA. San Francisco, August 2017.

Katharina Mucha. Ein Simulationsansatz zur Bewertung von Hitzestressrisiken in Innenräumen. Weiterentwicklung eines zonalen Modells in Modelica. Dissertation, Fakultät Gestaltung, Universität der Künste Berlin, 2017.

pythonOCC. pythonOCC – 3D CAD for python - http://www.pythonocc.org (last access on 2017 Jan 20).

QGIS. Ein freies Open-Source-Geographisches-Informationssystem - http://www.qgis.org/de/site (last access on 2017 Jan 20).

Senatsverwaltung für Stadtentwicklung und Wohnen: Sanierungsgebiet Friedrichshain-Kreuzberg – Rathausblockhttp://www.stadtentwicklung.berlin.de/staedtebau/foerderprogramme/stadterneuerung/de/rathausblock/index.shtml (last access on 2016 Dec 29).

Matthis Thorade, Jörg Rädler, Peter Remmen, Tobias Maile, Reinhard Wimmer, Jun Cao, Moritz Lauster, Christoph Nytsch-Geusen, Dirk Müller, and Christoph van Treeck. An open toolchain for generating Modelica code from Building Information Models. *11th International Modelica Conference*, p.383–391, Versailles, September 2015.

UCaHS. DFG Research Unit 1736 UCaHS - Urban Climate and Heat Stress in mid-latitude cities in view of climate change http://www.ucahs.org (last access on 2017 Jan 20).

Wetter Michael, Fuchs Marcus, Grozman Pavel, Helsen Lieve, Jorissen Filip, Lauster Moritz, Müller Dirk, Nytsch-Geusen Christoph, Picard Damien, Sahlin Per, and Thorade Matthis. IEA EBC Annex 60 Modelica Library - An international collaboration to develop a free open-source model library for buildings and community energy systems. 14th IBPSA Building Simulation Conference, Hyderabad, Conference Proceedings, December 2015.